

REMARKS

In response to the Office Action mailed July 30, 2004, Applicant respectfully requests reconsideration. To further the prosecution of this Application, Applicant submits the following remarks and has added new claims. The claims as now presented are believed to be in allowable condition.

Claims 1-30 were pending in this Application. By this Amendment, claims 31-34 have been added. Accordingly, claims 1-34 are now pending in this Application. Claims 1, 7, 14 and 21 are independent claims.

Preliminary Matters

Applicant has made note that Applicant's earlier amendment to the Specification has been entered. Applicant wishes to thank Examiner Fleurantin for withdrawing the objection to the Specification in view of that earlier amendment.

Rejections under §103

Claims 1-12, 14-19 and 21-30 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,884,098 (Mason, Jr.) in view of U.S. Patent No. 6,581,063 (Kirkman). Claims 13 and 20 were rejected under 35 U.S.C. §103(a) as being unpatentable over Mason, Jr. in view of European Patent 0114190 A2 (Hartung).

Applicant respectfully traverses each of these rejections and requests reconsideration. The claims are in allowable condition because that patentably distinguish over the cited prior art.

Mason, Jr. discloses a RAID disk controller 201 having a front end cache 205 and a back end cache 209 (column 5, lines 6-24 and Fig. 2). The front end cache 205 and the back end cache 209 are separate software processes executing on one or more microcontrollers which exercise control over all of the disk controller 201 (column 5, lines 24-36). Although only one cache memory is used, the front end cache 205 and the back end cache 209 may either share a

single control store or use separate control stores to hold control structures such as a least recently used (LRU) block queue (column 6, lines 7-11). For convenience, the following description refers to the queue of blocks available in each cache system simply as a front end cache block list and back end cache block list (column 6, lines 11-13). LRU queue algorithms and techniques are well known (column 6, line 14). In particular, an LRU cache block list may be a data structure configured as a doubly linked list holding a pointer to each cache block currently allocated in the cache memory (column 6, lines 14-18). When a block which is already in the queue is used, the entry for that block in the list is moved to the head of the list (column 6, lines 18-20). The entry in the list corresponding to the block which is used at the earliest time (or even never used) eventually moves to the last position in the list (column 6, lines 20-23). When the cache memory becomes full, subsequent attempts to allocate cache blocks cause the last entry in the list to be removed, deallocating the cache block pointed to (column 6, lines 23-26). The cache memory block previously pointed to by the last entry in the list is then replaced with the new block for which allocation was attempted and an entry for the new block placed at the head of the list (column 6, lines 26-29). To optimize the performance of this structure, it is desirable to coordinate the operations performed by the front end cache 205 and the back end cache 209 (column 6, lines 30-33). In particular, there is a communication path 211 established between the front end cache 205 and the back end cache 209 (column 6, lines 52-55). The communication path 211 established between the front end cache 205 and the back end cache 209 may take several forms including communications hardware built into the disk array controller 201 (e.g., one microprocessor controlled by a second microprocessor over a dedicated serial line, two processors merged into a single hardware entity performing both functions in response to a multitasking software system, a system bus or shared memory (column 6, line 60 through column 7, line 12). The presently preferred communication path 211 is a common data structure residing in a common

control store accessible to both the front end cache 205 and the back end cache 209 (column 7, lines 13-16).

Kirkman discloses that a linked list has an associated auxiliary data structure, containing external references to the linked list which are used by updating tasks (column 2, lines 52-55). The auxiliary data structure is used to block subsets of the linked list from being altered by other tasks, thus allowing concurrent updates of discrete subsets (column 2, lines 55-57). The auxiliary data structure is an object called SharedList (column 2, lines 58-61). In operation, a mutator object blocks off a portion of the linked list by altering pointers so that point to a blocker object, the blocker object containing additional pointers by-passing the blocked list portion (column 3, lines 5-8). Other inspectors may thereafter traverse the list through the by-passed portion, although mutators traversing a list wait at the block portion (column 3, lines 8-11). By blocking only a relatively small portion of a large linked list, multiple tasks may concurrently access different list elements and perform separate list updates, thus improving performance, particularly where multiple processors are used (column 3, lines 12-16). Specifically, by blocking at the level of individual list elements as opposed to the entire list, multiple mutators may simultaneously traverse and alter different parts of a linked list without interfering with each other, and inspectors may concurrently traverse a list being altered by one or more mutators (column 3, lines 18-21). Within operating system 201, within database 202, and within application 203, are linked list data structures 210, 211, 212 containing linked lists of data elements (column 4, lines 64-67 and Fig. 2). Each linked list data structure 210, 211, 212 further contains auxiliary data structures used for accessing and updating the data in the linked list (column 5, lines 10-14). The SharedList object provides a means of maintaining a doubly-linked list of client objects, but with some unique characteristics, i.e., (a) clients may safely traverse through the list while links are being added to it and/or removed from it; and (b) multiple clients may independently and concurrently add or remove links to/from the list (column 6, lines 4-9). SharedListIterator<T>

objects 401 provide the ability to traverse ("walk through") the links of a SharedList<T> object 301, and to insert or remove links into/from the list (column 7, lines 60-64).

Hartung discloses methods and apparatus for peripheral data handling hierarchies (Title). In connection with data processing systems manufactured by IBM, there is a term "channel commands" (page 12, lines 1-3). A user identification (USEID) accompanies the ESTABLISH REFERENCE CHARACTERISTICS (ERC) command (page 12, lines 6-15). When erasure of USEIDs indicate detection of a control error, then a DCB 61 USEID section is examined (page 12, lines 21-36). Additionally, a parity error indicates an error in control data structures, and other error detection codes may also be employed (page 26, lines 4-5).

Claims 1-6 and 23-24

Claim 1 is directed to a memory board for a data storage system. The memory board includes an interface which is configured to couple to a bus of the data storage system, memory which is configured to store a doubly linked list data structure, and a memory board control circuit coupled to the interface and the memory. The memory board control circuit is configured to receive a modify command from a processor of a data storage system through the interface. The processor is configured to move data within the data storage system. The memory board control circuit is further configured to atomically modify the doubly linked list data structure in accordance with the modify command, and provide a result to the processor of the data storage system through the interface in response to modifying the doubly linked list data structure.

The cited prior art does not teach or suggest a memory board for a data storage system which includes a memory board control circuit where the memory board control circuit is configured to atomically modify the doubly linked list data structure in accordance with a modify command, as recited in claim 1. Along these lines, the Office Action states at the bottom of page 4 that Mason does not

disclose the claimed "atomically modify the doubly linked list data structure in accordance with a modify command." Applicant agrees.

However, the Office Action then contends on page 5, lines 7-10 that it would have been obvious to one having ordinary skill in the art at the time the invention was made to modify the combined teachings of Mason and Kirkman with "atomically modify the doubly linked list data structure in accordance with a modify command." Applicant respectfully disagrees with this contention.

In an attempt to support this contention, the Office Action on page 5, lines 10-12 alleges that "such modification would allow the teachings of Mason and Kirkman to provide the ability to traverse the links of a Share list object and to insert or remove links into/from the list, (see Kirkman col. 7, lines 60-62)." **This allegation is incorrect and the cited portion of Kirkman provides no such statement.** To the contrary, Kirkman, in column 7, lines 60-62 states that "SharedListIterator<T> objects 401 provide the ability to traverse ("walk through") the links of a SharedList<T> object 301, and to insert or remove links into/from the list." Accordingly, Kirkman clearly explains that the ability to traverse and insert/remove links in the Kirkman system is achievable, but there is no mention about atomically modifying whatsoever in the cited passage. Therefore, the Kirkman system already provides the ability to traverse the links without atomically modifying, and the allegation of the Office Action is wrong.

Additionally, the Office Action at the top of page 5 states that Kirkman discloses

a means of maintaining a doubly-linked list of client object, but with some, i.e., unique characteristics clients may safely traverse through the list while links are being added to it and/or removed from it; and multiple clients may independently and concurrently add or remove links to/from the list. The Shared List object allows all clients to access the list simultaneously. It provides blocking of tasks at the link level and only those tasks involved in the individual link conflicts are involved in blocking, (see Kirkman, col. 6, lines 6-13) and see col. 4, line 64 to col. 5, line 32.
Sic.

Applicant respectfully submits that regardless of whether this contention is true, this still is not a teaching of atomically modifying a doubly linked list data structure in accordance with a modify command, as recited in claim 1.

For a proper rejection under 35 USC §103 and to establish a *prima facie* case of obviousness, the Office Action must meet three criteria.

“First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations.”¹

Applicant respectfully submits that the Office Action does not meet all of these three criteria. For example, the prior art references do not teach or suggest all the claim limitations since there is no atomically modifying disclosed by the combination of Mason and Kirkman.

For the reasons stated above, claim 1 patentably distinguishes over the cited prior art, and the rejection of claim 1 under 35 U.S.C. §103(a) should be withdrawn. Accordingly, claim 1 is in allowable condition.

Because claims 2-6 and 23-24 depend from and further limit claim 1, claims 2-6 and 23-24 are in allowable condition for at least the same reasons. Additionally, it should be understood that the dependent claims recite additional features which further patentably distinguish over the cited prior art.

Claims 7-13 and 25-26

Claim 7 is directed to a data storage system having a set of storage devices, a processor which is configured to move data to and from the set of storage devices, a bus coupled to the processor, and a memory board. The memory board includes (i) an interface which couples to the bus, (ii) memory

¹ *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991).

which is configured to store a doubly linked list data structure, and (iii) a memory board control circuit coupled to the interface and the memory. The memory board control circuit is configured to receive a modify command from the processor of the data storage system through the interface and the bus, atomically modify the doubly linked list data structure in accordance with the modify command, and provide a result to the processor of the data storage system through the interface and the bus in response to modifying the doubly linked list data structure.

As explained above in connection with claim 1, the cited prior art does not teach or suggest such a memory board control circuit. In particular, Applicants submit that the Office Action (middle of page 11) provided an invalid argument for claim 7 which was identical to that submitted in connection with claim 1. Accordingly, the cited prior art also does not teach a data storage system having such a memory board control circuit. As a result, claim 7 patentably distinguishes over the cited prior art for at least the same reasons as claim 1. Therefore, the rejection of claim 7 under 35 U.S.C. §103(a) should be withdrawn, and claim 7 is in allowable condition.

Because claims 8-13 and 25-26 depend from and further limit claim 7, claims 8-13 and 25-26 are in allowable condition for at least the same reasons.

Claims 14-20

Claim 14 is directed to a method for accessing a doubly linked list data structure. The method is performed in a memory board of a data storage system and includes the step of receiving a modify command from a processor of a data storage system through a bus of the data storage system. The processor is configured to move data within the data storage system. The method further includes the steps of atomically modifying the doubly linked list data structure in accordance with the modify command, and providing a result to the processor of the data storage system through the bus in response to modifying the doubly linked list data structure.

As explained above in connection with claim 1, the cited prior art does not teach or suggest such a method because the cited prior art does not disclose atomically modifying a doubly linked list data structure in accordance with a modify command. In particular, Applicants submit that the Office Action (top of page 17) provided an invalid argument for claim 14 which was identical to that submitted in connection with claim 1. Accordingly, the cited prior art also does not teach a method having such a step. As a result, claim 14 patentably distinguishes over the cited prior art for at least the same reasons as claim 1. Thus, the rejection of claim 14 under 35 U.S.C. §103(a) should be withdrawn, and claim 14 is in allowable condition.

Because claims 15-20 and 27-28 depend from and further limit claim 14, claims 15-20 and 27-28 are in allowable condition for at least the same reasons.

Claims 21-22

Claim 21 is directed to a memory board control circuit for accessing a doubly linked list data structure of a data storage system. The memory board control circuit is mountable to a memory board. The memory board control circuit includes an input port that couples to a bus of the data storage system, an output port that couples to the bus of the data storage system, and control logic connected to the input port and to the output port. The control logic is configured to receive a modify command from a processor of a data storage system through the input port. The processor is configured to move data within the data storage system. The control logic is further configured to atomically modify the doubly linked list data structure in accordance with the modify command, and provide a result to the processor of the data storage system through the output port in response to modifying the doubly linked list data structure.

As explained above in connection with claim 1, the cited prior art does not teach or suggest a memory board control circuit configured to atomically modify a doubly linked list data structure in accordance with a modify command. In particular, Applicants submit that the Office Action (bottom of page 23) provided

an invalid argument for claim 21 which was identical to that submitted in connection with claim 1. Accordingly, claim 21 patentably distinguishes over the cited prior art for at least the same reasons as claim 1. Thus, the rejection of claim 21 under 35 U.S.C. §103(a) should be withdrawn, and claim 21 is in allowable condition.

Because claims 22 and 29-30 depend from and further limits claim 21, claims 22 and 29-30 are in allowable condition for at least the same reasons.

Newly Added Claims

Claims 31-34 have been added and are believed to be in allowable condition. Claim 31 depends from claim 24. Claim 32 depends from claim 26. Claim 33 depends from claim 28. Claim 34 depends from claim 30. Support for claims 31-34 is provided within the Specification, for example, on page 5, line 24 through page 6, line 23. No new matter has been added.

Conclusion

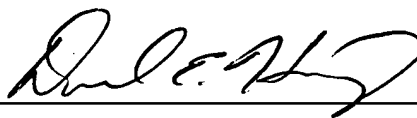
In view of the foregoing remarks, this Application should be in condition for allowance. A Notice to this affect is respectfully requested. If the Examiner believes, after this Amendment, that the Application is not in condition for allowance, the Examiner is respectfully requested to call the Applicant's Representative at the number below.

Applicant hereby petitions for any extension of time which is required to maintain the pendency of this case. If there is a fee occasioned by this Amendment, including an extension fee, that is not covered by an enclosed check, please charge any deficiency to Deposit Account No. 50-0901.

-21-

If the enclosed papers or fees are considered incomplete, the Patent Office is respectfully requested to contact the undersigned collect at (508) 366-9600, in Westborough, Massachusetts.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "David E. Huang", is written over a horizontal line.

David E. Huang, Esq.
Attorney for Applicant
Registration No.: 39,229
CHAPIN & HUANG, L.L.C.
Westborough Office Park
1700 West Park Drive
Westborough, Massachusetts 01581
Telephone: (508) 366-9600
Facsimile: (508) 616-9805

Attorney Docket No.: EMC01-19(01056)

Dated: November 1, 2004